UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Diseño e implementación de un gestor de contraseñas respaldado en la nube

Autor: Eduardo Hilario Serrano
Tutor: Óscar Delgado Mohatar
Ponente: Eloy Anguiano Rey

junio 2019

Eduardo Hilario Serrano
Diseño e implementación de un gestor de contraseñas respaldado en la nube

Eduardo Hilario Serrano
C\ Francisco Tomás y Valiente Nº 11

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

A mi madre, mi padre y mis hermanos

Nunca confíes en un ordenador que
no puedas lanzar por una ventana.

Steve Wozniak

Agradecimientos

Me gustaría agradecer a todas las personas que me han apoyado en algún momento de mi vida hasta la realización de este trabajo.

En especial a mi familia, entre ellos: a mi madre, por acompañarme los momentos más difíciles; a mi padre, por su ayuda cuando la he necesitado y a mis hermanos, por estar siempre ahí.

También a mis amigos, Daniel, Mario y Javier; por su complicidad, camaradería y abnegación.

Por último, a los profesores que me han acompañado a lo largo de la carrera, con especial mención a Idoia Alarcón, por preocuparse por mí; Óscar Delgado, al que le debo más agradecimientos aparte de haberse ofrecido a acompañarme en este trabajo y Eloy Anguiano, entre otras cosas, por la clase de LaTeX que ha ayudado a escribir esta memoria.

A todos vosotros, gracias.

Resumen

El objetivo de este proyecto es el desarrollo de una aplicación de gestión de contraseñas auto contenida que pueda ser ejecutada sin conexión a Internet en la que el usuario tenga el control real de sus datos y se respete su privacidad, sin que tenga obligación de ceder ningún tipo de información.

El proyecto se ha implementado teniendo en cuenta siempre que ha sido necesario, la seguridad por encima de cualquier otro requisito, priorizándolo por ejemplo sobre la usabilidad, aunque ésta tampoco ha sido descuidada. Por ello los algoritmos de cifrado se han utilizado con parámetros recomendados para la protección de información crítica y se ha diseñado teniendo en mente un escenario de ataque en el que se tenga acceso total al sistema.

Como herramientas principales para el desarrollo, se ha usado el framework Electron.js, para obtener una aplicación funcional en los tres principales sistemas operativos de escritorio empleando HTML, CSS y JavaScript. Se ha utilizado de Node.js, sus APIs de criptografía y para el almacenamiento de información, SQLite, por sus características como resultar auto contenido y sin servidor.

Palabras clave

Cifrado, Seguridad, Protección de información, aplicación de escritorio.

Abstract

The main purpouse of this proyect is the development of an application for password management self-contained that could be executed without Internet connection and where the user have trust power over his data and the privacity be respected, without any obligation to give any tipe of information.

The proyect has been implemented always taking into account when have been needed, the security over any other requirement, prioritizing it for example over the usability, although this one either it has been never neglected. Because of that the encryption algorithms have been used with parameters recommended for protect critical information and it has been designed keeping in mind an intrusion scene where you have full access to the system.

As main development tools, we have been used Electron.js framework, to get an functional application into the three leading desktop operative systems using HTML, CSS and JavaScript. We habe been used Node.js too, it's APIs of cryptography and for the data storage, SQLite, because of his features as self-contained and serverless.

Keywords

Encryption, Security, Information protection, desktop application.

Índice

1 Introducción **1**

1.1 Marco del proyecto 1

1.2 Motivación 2

1.3 Objetivos 2

1.4 Estructura de la memoria 2

2 Estado del arte **5**

2.1 Introducción 5

2.2 Extensiones para navegadores Web 5

2.3 Programas Locales 6

2.3.1 Conexión a internet necesaria 6

2.3.2 Aplicaciones auto contenidas 7

3 Análisis **9**

3.1 Introducción 9

3.2 Métodos de cifrado 9

3.2.1 Cifrado de clave simétrica 9

3.2.2 Derivación de claves 13

3.3 Roles de usuario 14

3.4 Casos de Uso 14

3.5 Requisitos 15

3.5.1 Requisitos Funcionales 15

3.5.2 Requisitos No Funcionales 16

3.6 Tecnologías para el desarrollo 16

3.6.1 Front-End 17

3.6.2 Back-End 17

3.7 Conclusiones del análisis 18

4 Diseño y Desarrollo **19**

4.1 Introducción 19

4.2 Diagrama de Bloques 19

4.3 Autenticación de segundo factor 20

4.4 Back-End 20

4.4.1 Base de Datos (SQLite) 20

4.4.2 Módulo de cifrado 22
4.4.3 Módulo de respaldo 22
4.5 Front-End 24
4.5.1 Interfaz 24
5 Pruebas y resultados **27**
5.1 Introducción 27
5.2 Funcionalidad 27
5.3 Usabilidad 27
5.4 Compatibilidad 28
6 Conclusiones y trabajo futuro **29**
6.1 Conclusiones 29
6.2 Trabajo Futuro 30
Bibliografía **31**
Definiciones **33**
Acrónimos **35**
Apéndices **37**
A Flujo de trabajo de un usuario para registrar datos **39**

Listas

Lista de figuras

3.1 Cifrado simétrico 10
3.2 Red Permutación-sustitución 11
3.3 Cifrado ECB 12
3.4 Cifrado CBC 13
3.5 PBKDF2 14
3.6 Casos de uso de la aplicación 15

4.1 Diagrama Bloques Proyecto 19
4.2 Arquitectura SQLite 21
4.3 Tablas de la base de datos 21
4.4 Flujo de OAuth2 23
4.5 Pantalla de Login 24

A.1 Inicio de la aplicación 39
A.2 Panel de Control 39
A.3 Inserción de datos 40
A.4 Panel actualizado 40
A.5 Visualización de información 41

1

Introducción

1.1. Marco del proyecto

En la época actual, los sistemas informáticos y la información cobran cada vez más relevancia. A día de hoy, es común que un profesional requiera de una o varias herramientas tecnológicas para desarrollar su actividad. Esta integración, llega hasta el punto que es normal encontrarse ya con dependencias críticas de estos dispositivos. Por ejemplo, las entidades bancarias a día de hoy no pueden permitirse la desconexión, aunque sea momentánea, de sus sistemas.

Cada vez son más complejos, lo cual muchas veces implica la necesidad de construir sistemas distribuidos y es común que el uso de muchas aplicaciones requiera de conexión a internet. Esto se traduce en que, por un lado, la cantidad de puntos de fallo aumenta; y por otro, la funcionalidad del sistema ya no depende únicamente de nuestro dispositivo.

Por tanto, ya no disponemos del control de las herramientas que son necesarias, incluso vitales, para nuestros objetivos e intereses y dependemos de terceros. Básicamente, nuestra información puede verse comprometida sin mucha dificultad, sobretodo en los casos en los que las conexiones entre las distintas partes se realiza mediante Internet.

Esto hace necesario, que todas las conexiones, sobretodo cuando se trata de información sensible, se realicen con las medidas de seguridad apropiadas y que la información, se almacene cifrada, para que en caso verse comprometida no resulte sencillo su acceso y uso por parte de terceros. Para ello, se hace necesario el uso de credenciales, contraseñas u otras formas de verificación de identidad para poder acceder a nuestra información. Además, para mayor seguridad, se recomienda y conviene, utilizar distintas contraseñas para cada servicio que necesitemos, por si se diera el caso de que uno de ellos se viera comprometido, el resto siguieran a salvo.

Sin embargo, teniendo en cuenta la cantidad de servicios que usamos a día de hoy, es realmente complicado recordar todos los credenciales de cada uno de los sistemas que utilizamos. Para solucionarlo, surgen utilidades, conocidas como gestores de contraseñas, cuyo servicio consiste es servir de llavero a sus usuarios.

1.2. Motivación

La razón que motiva la realización de este proyecto atiende a la oferta actual de gestores de contraseñas existentes. La mayoría de ellos son gestores *online*, los cuales requieren de una conexión a Internet, por lo que el usuario necesita estar conectado para su uso y además depende del correcto funcionamiento del servicio el cual en un momento dado puede no responder y no permitir el acceso a los datos.

El principal problema que ha motivado el trabajo con respecto a estos sistemas, atiende a la realidad de que los datos no dependen del usuario, sino de terceros y en entornos en los que dichas contraseñas sean críticas, esto puede no resultar una condición aceptable.

Con respecto a los sistemas *offline*, no hemos encontrado sistemas que cumplieran todas las capacidades que buscábamos al mismo tiempo, entre ellas: ser multiplataforma, permitir respaldos automáticos en la nube, ser de código abierto, etcétera.

1.3. Objetivos

El objetivo de este proyecto es desarrollar una aplicación de escritorio totalmente funcional que permita la gestión de contraseñas por parte del usuario garantizando su seguridad haciendo uso de varios métodos de cifrado que permitan asegurar la integridad de los datos ante un escenario en el que un atacante tenga acceso total a todo el sistema del usuario.

Además se plantea que la aplicación sea sencilla de utilizar, compatible con los principales sistemas operativos de escritorio y permita al usuario vincular una solución de nube o almacenamiento *online* para tener la posibilidad de guardar los datos respaldados en caso de que el dispositivo del usuario falle.

Para ello, se han planteado los siguientes hitos:

- Estudiar y analizar los diferentes métodos de cifrado de clave simétrica y generación de claves.
- Analizar las tecnologías que permitan el desarrollo de la aplicación con las características requeridas.
- Diseñar y desarrollar la aplicación haciendo uso de los métodos de cifrado y tecnologías que resulten más convenientes.
- Realizar pruebas del producto obtenido para verificar que se ajusta a los requisitos planteados.

1.4. Estructura de la memoria

La memoria consta de los siguientes capítulos:

- En el **capítulo 2**, se mencionan las soluciones que se han encontrado existentes en la actualidad.

- En el **capítulo 3**, se describen los métodos de cifrado y tecnologías para el desarrollo del proyecto. También se decide como se va a diseñar la aplicación, los casos de uso de ésta y la definición de requisitos.
- En el **capítulo 4**, se trata cómo se ha realizado el diseño y desarrollo de la aplicación además de cómo se han integrado las tecnologías usadas en el proyecto.
- En el **capítulo 5**, se discute el resultado del desarrollo y se comentan las pruebas realizadas.
- En el **capítulo 6**, se expresan las conclusiones a las que se ha llegado con este trabajo y se detallan las posibles líneas de trabajo futuro que se pudieran llevar a cabo como continuación del proyecto.

2

Estado del arte

2.1. Introducción

Este proyecto de Trabajo de Fin de Grado (TFG) tiene el fin de crear una aplicación de gestión de contraseñas auto contenida, con un alto estándar de seguridad, permitiendo realizar respaldos automáticos y accesible a todo el mundo.

Para ello, en primer lugar, se detalla un estado del arte enfocado en conocer una selección de aplicaciones y servicios que funcionan como gestores de contraseñas entre los que se han estudiado que se han considerado como los más relevantes, y que constituyen una buena muestra de la oferta actual, por sus aspectos técnicos para entender qué aportan y decidir el enfoque del proyecto.

2.2. Extensiones para navegadores Web

Son servicios que se ejecutan como complementos de un navegador web, su uso queda normalmente limitado, por tanto, a servicios web usando un dispositivo y navegador concretos.

LastPass

Esta extensión [1], está disponible para los principales navegadores. Funciona mediante una contraseña maestra que usa los cifrados Advanced Encryption Standard (AES) de 256 bits [2] y Password-Based Key Derivation Function 2 (PBKDF2) con Secure Hash Algorithm (SHA) de 256 bits [3]. El usuario tiene acceso dentro del navegador a un baúl donde se guardan sus contraseñas y también, si lo desea, más tipos de información como tarjetas bancarias o documentos.

Permite sincronización entre varios dispositivos, Autenticación de segundo factor (2FA) e incluso tiene algunas características de pago que permiten compartir datos con otros usuarios.

Cuando se accede a un servicio para registrarse o entrar en éste, la extensión detecta la página y guarda o escribe la contraseña.

DashLane

DashLane [4], está disponible como extensión además de tener programa para Windows y aplicaciones para sistemas operativos móviles. Permite guardar documentos, no solo contraseñas, las cuales reconoce automáticamente en las páginas web tanto para registrarlas como para rellenarlas.

También realiza una monitorización de la Dark Web para informar a los usuarios de forma personalizada de posibles alertas de filtraciones de datos. Además sincroniza todos los datos entre dispositivos.

2.3. Programas Locales

Se trata de aplicaciones que se instalan en el dispositivo del usuario, aunque podemos diferenciar dos tipos: las que necesitan de conexión a Internet para funcionar y las que no.

2.3.1. Conexión a internet necesaria

En estos casos usualmente todos los datos se suelen guardar en servidores pertenecientes a los propietarios del servicio y la aplicación sirve para conectarse a ellos. Por tanto, no se diferencian demasiado de las extensiones de navegador comentadas desde un punto de vista de la ubicación de los datos.

1Password

Probablemente la aplicación más completa de las que hemos analizado y de las que se encuentran actualmente en el mercado [5]. No obstante, es una aplicación de pago aunque disponga de una prueba gratuita y guarda los datos en almacenamiento *online*, no en local. Concretamente usan Amazon Aurora, uno de los servicios de Amazon Web Services (AWS) que encapsula una base de datos MySQL relacional en la que almacenan la información en una estructura JavaScript Object Notation (JSON) en la que los campos correspondientes a los datos sensibles son información binaria comprimida y cifrada.

Para cifrar los datos usan una contraseña maestra que debe ser custodiada por el usuario y una clave secreta que es un identificador generado y guardado en local de 128 bits a los que le aplican una derivación de claves basada en Código de Autenticación de Mensajes en clave-hash (HMAC) y PBKDF2 usando SHA de 256 bits con 100.000 iteraciones.

Disponen también de un libro blanco que explica en detalle el diseño de seguridad de su sistema lo cual permite que terceros comprueben su robustez y ofrecen recompensas a cambio de fallos de seguridad que sean encontrados.

2.3.2. Aplicaciones auto contenidas

Por último, se han estudiado aplicaciones con un enfoque similar al que se quiere dar en este proyecto, que el usuario no dependa de ningún sistema externo a su dispositivo para poder acceder a sus datos.

Password Safe

Se trata de una aplicación de escritorio Open Source compatible con la mayoría de plataformas que guarda los datos cifrados en una base local [6]. Para cifrarlos utiliza el algoritmo *Twofish* con una clave de 256 bits. Este algoritmo fue uno de los cinco finalistas a convertirse en el AES .

Tiene una interfaz sencilla y permite organizar los datos por categorías.

KeePassXC

Basado en KeePassX, a su vez basado en KeePass; se trata de una aplicación de escritorio Open Source con versión para Windows, Linux y MacOS [7]. Permite elegir el algoritmo de cifrado de los datos entre AES , *Twofish* y *ChaCha20*. Utiliza un formado de base de datos propio para guardar la información y dispone de integración con navegadores web.

Tiene algunas características adicionales interesantes como interfaz para línea de comandos o que permite conexión mediante Secure Shell (SSH) .

3 Análisis

3.1. Introducción

En este apartado se va a realizar un análisis de los distintos aspectos a considerar para el desarrollo del proyecto. Empezaremos con un análisis de los métodos de cifrado que nos pueden resultar útiles, siguiendo con los roles de usuario, casos de uso, requisitos, tecnologías para el proyecto y por último las conclusiones del análisis.

La aplicación a desarrollar recibirá el nombre de UnmissablePasswords.

3.2. Métodos de cifrado

Para poder guardar la información de los usuarios de la aplicación con seguridad es necesario aplicar métodos y algoritmos que permitan almacenar dicha información de manera que un tercero no pueda hacer nada útil con ella a no ser que se trate del propio usuario.

Dado que no es necesario en este caso compartir información con otra persona mediante un canal no seguro, sino que simplemente se trata del propio usuario que accede a su propia información; el método a utilizar por la aplicación será el cifrado de clave simétrica. Además para dar mayor seguridad a la clave utilizada, se utilizará la derivación de claves para reforzarla.

3.2.1. Cifrado de clave simétrica

El cifrado de clave simétrica consiste en que la información en claro se cifra mediante una clave utilizando un algoritmo generando a partir de ello una información cifrada. Para poder utilizar dicha información es necesario aplicar el mismo algoritmo con la misma clave sobre la información cifrada y de esta forma se vuelve a obtener la información en claro. Un esquema de este proceso se puede observar en la figura 3.1.

Es importante destacar que el algoritmo que se use para cifrar la información puede perfectamente

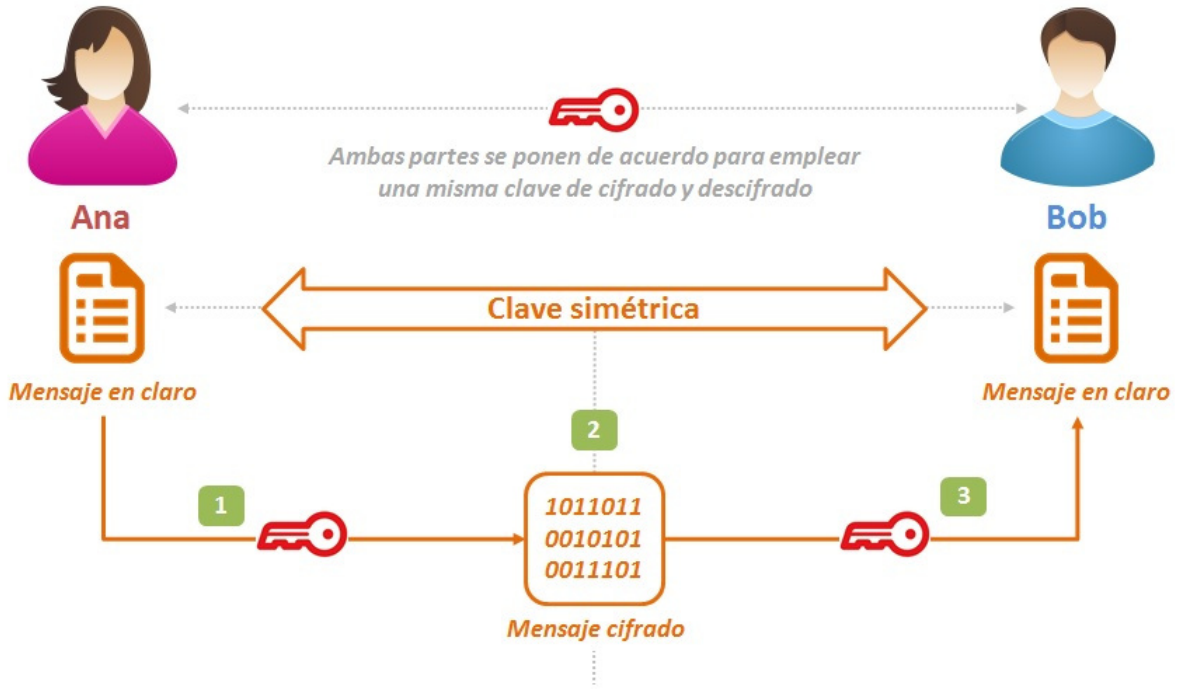

Figura 3.1: Diagrama de funcionamiento del cifrado simétrico.
Fuente: https://securitcrs.wordpress.com

y sin ningún problema ser público, mientras que la clave usada para el cifrado debe custodiarse con precaución, ya que de ella depende el acceso a la información en claro.

Probablemente el algoritmo más importante de cifrado simétrico usado a día de hoy sea AES , el cual veremos a continuación.

Advanced Encryption Standard (AES)

Este algoritmo, también conocido como *Rijndael* se convirtió en AES después de ganar en 1997 el concurso organizado por el Instituto Nacional de Normas y Tecnología (NIST) de Estado Unidos para escoger un algoritmo de cifrado que fuera robusto por lo menos durante el siglo XXI. Como características interesantes de AES podemos enumerar las siguientes:

- Tiene un tamaño de bloque fijo de 128 bits.
- La clave puede ser de 128, 192 o 256 bits.
- Es de dominio público.
- Es rápido y sencillo de implementar tanto en software como en hardware.
- Requiere poca memoria.

Centrándonos en la longitud de clave, de ésta depende la seguridad del algoritmo, puesto que 2 elevado a la cantidad de bits de la clave son las posibles claves de cifrado que existen y esta cantidad de posibilidades es la que nos protege de ataques de fuerza bruta. Si pensamos en ello, la probabilidad

de que una persona acierte la clave de cifrado usada teniendo una longitud de 256 bits, es virtualmente imposible y un ataque de fuerza bruta suponiendo que se encuentra la clave correcta tras probar el 50 % de las combinaciones posibles, nos llevaría con la tecnología actual más tiempo que años se estima que tiene el universo [8].

Las dos últimas cualidades del algoritmo son debidas a que el algoritmo está basado en una red de sustitución-permutación, como indica la figura 3.2.

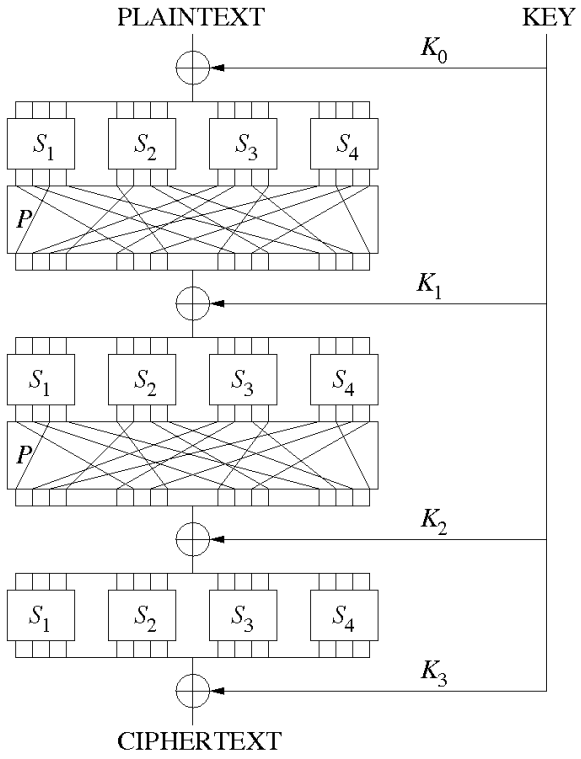

Figura 3.2: Esquema de una red de permutación-sustitución
Fuente: Encyclopedia of Cryptography and Security; Henk C. A. van Tilborg, Sushil Jajodia

El algoritmo opera sobre una matriz de 4x4 bytes y tiene tres fases:

1.– Fase inicial

   1.1.– Combinamos cada byte de la matriz con un byte de la subclave mediante una operación XOR (*AddRoundKey*).

2.– Fase central (se repite 10 veces para claves de 128 bits, 12 para 192 y 14 para 256)

   2.1.– Sustituimos cada byte por otro mediante una tabla de búsqueda no lineal (*SubBytes*).

   2.2.– Rotamos cada fila un número determinado y distinto de veces (*ShiftRows*).

   2.3.– Mezclamos los bytes de cada columna usando una transformación lineal (*MixColumns*).

   2.4.– Aplicamos el paso *AddRoundKey* explicado anteriormente.

3.– Fase final

   3.1.– Aplicamos el paso *SubBytes*.

   3.2.– Aplicamos el paso *ShiftRows*.

3.3.– Aplicamos el paso *AddRoundKey*.

Cifrado por bloques

Como hemos podido apreciar en el desarrollo del algoritmo de cifrado AES , este cifra bloques de 128 bits. En las aplicaciones reales, no es lo normal que la información que se desee cifrar ocupe exactamente 128 bits. Por ello, el procedimiento habitual es dividir la información en bloques del tamaño requerido por el algoritmo de cifrado, rellenando parcialmente el último bloque con información vacía en el caso de que al realizar la división quedare incompleto. De esta forma podemos aplicar el algoritmo a cada bloque.

Sin embargo, si no hiciéramos nada más, si dos bloques fueran iguales, en el resultado global final tendríamos un problema de información semántica. Un posible atacante no podría saber exactamente qué información contiene el bloque cifrado pero el conjunto de bloques podría darle información acerca de la información cifrada. Este tipo de cifrado se conoce como Electronic Code-Book (ECB) y aunque pueda ser más rápido que el método que veremos a continuación debido a que el cifrado de bloques puede ser paralelizado, no es recomendable usarlo para proteger la información de nuestro gestor de contraseñas. En la figura 3.3 se aprecia claramente por qué no es buena idea usar ECB [9].

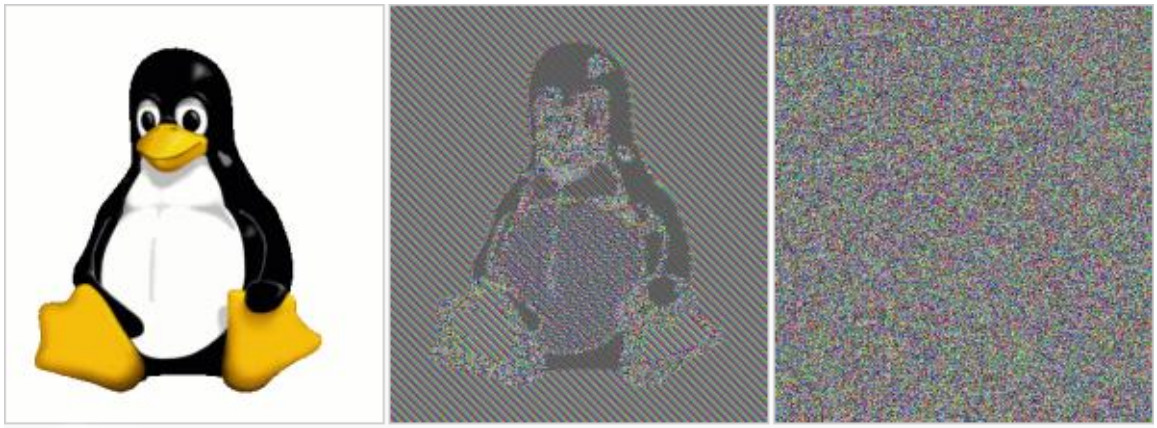

Figura 3.3: Imagen cifrada usando el modo ECB (centro) respecto a otros más sofisticados (derecha). Fuente: Universidad de Brown

Para solucionar este problema, se puede usar el modo Cipher-block chaining (CBC) , que consiste en aplicar a cada bloque antes de ser cifrado una operación XOR con el bloque cifrado anterior (Figura 3.4), usando un vector de inicialización aleatorio para el primer bloque [9]. Aunque tenga la desventaja de no poder ser paralelizado al depender el bloque siguiente del anterior, aplicar esta técnica aumenta sensiblemente la seguridad de la información.

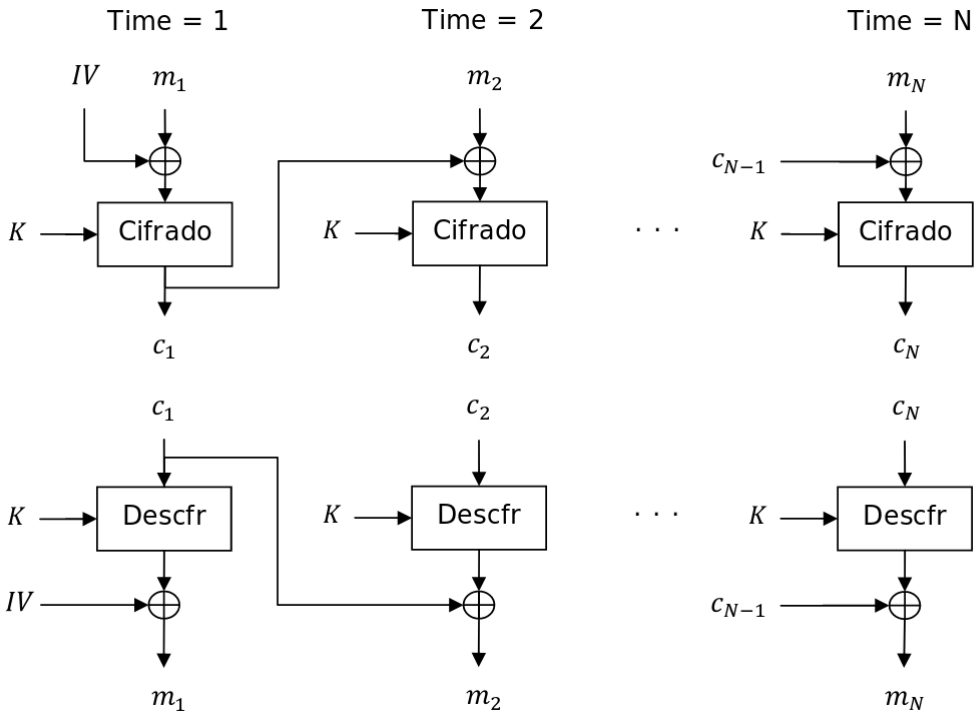

Figura 3.4: Esquema de aplicación del método CBC
Fuente: Universidad Politécnica de Cartagena

3.2.2. Derivación de claves

La derivación de claves mediante una función pseudoaleatoria sirve en criptografía para transformar contraseñas que pudieran no reunir las propiedades deseadas para ser usadas como claves de algoritmos de cifrado en otras que sí las posean. Principalmente aportan dos beneficios:

- Obtener claves de cifrado de la longitud deseada para el algoritmo de cifrado.
- Complicar sensiblemente los ataques de fuerza bruta o de diccionario.

Las funciones de derivación de clave funcionan normalmente con una clave de entrada, un salt aleatorio y una cantidad de iteraciones; a partir de lo cual devuelven una clave resultante y si los parámetros de la función permanecen invariantes, la clave resultante también será la misma.

Password-Based Key Derivation Function 2 (PBKDF2)

Esta función de derivación de claves [3] funciona a partir de cinco parámetros: una función pseudoaleatoria, una contraseña, un salt, una cantidad de iteraciones y una longitud final de clave. El funcionamiento de éste se puede observar en la figura 3.5.

Este método de derivación de claves tiene un inconveniente, ya que consume muy poca RAM y es posible implementarlo también como circuito hardware sencillo que funcione muy rápido. Por tanto,

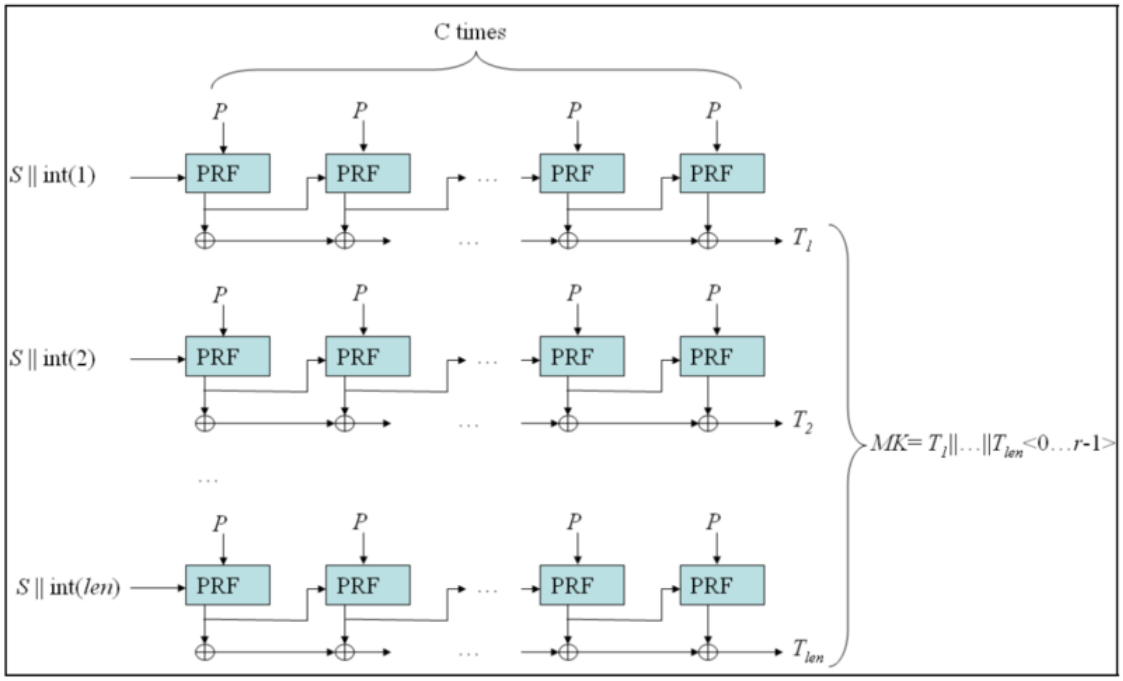

Figura 3.5: Esquema de funcionamiento de PBKDF2
Fuente: NIST - 800-132

si queremos asegurarnos de que no sea fácilmente vulnerable mediante ataques de fuerza bruta, es necesario realizar un número elevado de iteraciones.

3.3. Roles de usuario

El proyecto tiene la intención de ser usado únicamente por parte de un usuario final, por lo que éste es el único agente externo que interactúa con el sistema. El usuario es el encargado de crear una contraseña maestra, agregar y eliminar información y vincular servicios de almacenamiento *online* para tener respaldos de sus datos.

3.4. Casos de Uso

El proyecto que se desea desarrollar como aplicación de gestión de contraseñas debe ser sencillo y visual para el usuario, sin que exista una cantidad de elementos superior a la necesaria que entorpezca al usuario de su función principal y con la funcionalidad centrada en servir como gestor de contraseñas. El diagrama de casos de uso puede encontrarse en la figura 3.6.

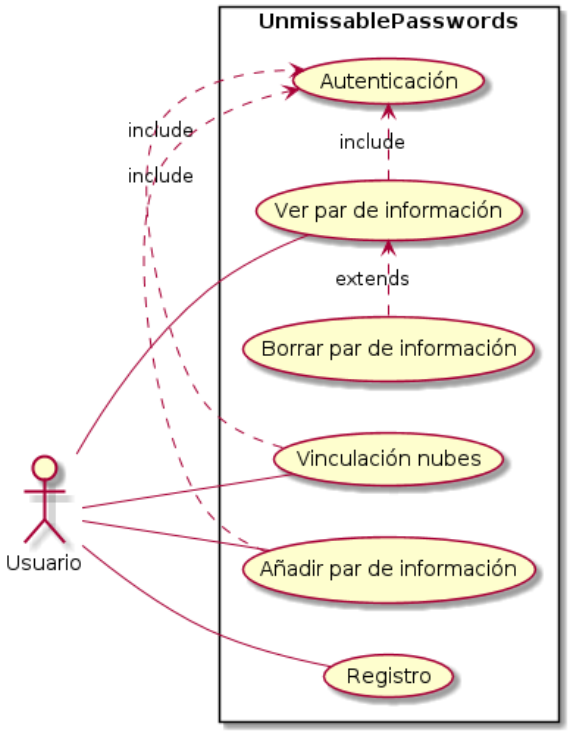

Figura 3.6: Diagrama de casos de uso de la Aplicación

3.5. Requisitos

Los requisitos se dividen en dos categorías:

- Requisitos funcionales
- Requisitos no funcionales

3.5.1. Requisitos Funcionales

A continuación se enumeran los requisitos funcionales que debe cumplir la aplicación:

RF-1.– El usuario podrá registrarse en el servicio mediante una contraseña.

RF-2.– El usuario podrá vincular una nube para respaldar sus datos.

RF-3.– El usuario podrá autenticarse en el servicio utilizando una contraseña.

RF-4.– El sistema guardará la información del registro cifrada.

RF-5.– El sistema deberá crear material criptográfico a partir de la contraseña del usuario.

RF-6.– El usuario podrá almacenar pares de información clave-valor.

RF-7.– El sistema mantendrá la información almacenada por el usuario cifrada.

RF-8.– El usuario podrá recuperar la información que almacene.

RF-9.– El usuario podrá eliminar de forma selectiva la información almacenada.

RF-10.– El sistema creará una estructura de respaldo de los datos cifrada.

RF-11.– Cuando sea posible, el sistema guardará la estructura de respaldo en la nube.

RF-12.– Cuando sea posible y necesario, el sistema recuperará la estructura de respaldo de la nube para restaurarlos.

3.5.2. Requisitos No Funcionales

Los requisitos no funcionales que se plantean para el diseño y desarrollo de la aplicación son los siguientes:

RNF-1.– El tiempo de aprendizaje y familiarización con la plataforma por parte de un usuario deberá ser menos de 15 minutos.

RNF-2.– Las operaciones que pueda realizar el usuario con la plataforma deberán completarse en tres pantallas o menos.

RNF-3.– La aplicación debe ser sencilla, es decir, no debe haber sobrecarga de elementos que distraigan la atención del contenido relevante.

RNF-4.– La aplicación debe ser intuitiva, es decir, los menús deben ser fáciles de encontrar y los distintos apartados deben estar descritos de forma clara y concisa.

RNF-5.– La información manejada por el sistema debe estar protegida contra accesos no autorizados.

RNF-6.– Se usarán únicamente protocolos de comunicación seguros para la transferencia de información.

RNF-7.– Toda funcionalidad del sistema, excepto las operaciones de cifrado que requieran un alto coste computacional, debe responder al usuario en menos de 2 segundos.

RNF-8.– Las operaciones de cifrado se realizarán en segundo plano para que el usuario pueda seguir usando el sistema.

RNF-9.– Se requiere de conexión a internet para el correcto funcionamiento del sistema.

RNF-10.– Se requiere que el usuario disponga de cuentas en servicios nube para realizar respaldos de la información.

RNF-11.– La aplicación deberá funcionar correctamente en los tres principales sistemas operativos de escritorio: Windows, Linux y MacOS.

RNF-12.– Deberá estar adaptada para múltiples dispositivos y tamaños de pantalla.

RNF-13.– Se priorizará la seguridad sobre la usabilidad.

3.6. Tecnologías para el desarrollo

El proyecto tiene como objetivo ser desarrollado como aplicación de escritorio que sea compatible con los principales sistemas operativos: Windows, Linux y MacOS. Como plantear un desarrollo específico en cada una de estas tres plataformas con herramientas nativas es muy costoso y tampoco se requiere hacer uso de utilidades específicas de cada plataforma; se ha buscado un framework que permita mediante el mismo código generar aplicaciones para las tres plataformas sin comprometer la seguridad.

Electron

Se trata de un framework creado inicialmente para el desarrollo del editor de texto Atom creado por GitHub y más tarde liberado para la construcción de cualquier aplicación, entre las que destacan Discord o Slack [10].

Está basado en Node.js [11] y Chromium [12] y aunque esto conlleve un coste importante a nivel computacional, compensa a nivel de desarrollo; dado que permite el uso de todos los módulos desarrollados para Node.js y la aplicación se puede desarrollar garantizando el mismo comportamiento en las tres plataformas para las que se quieren soportar mediante las mismas herramientas que usaríamos para un desarrollo web, HyperText Markup Languaje Version 5 (HTML5) , Cascade Style Sheet Version 3 (CSS3) y JavaScript.

Además dispone de un repertorio de APIs que permite el acceso a muchos recursos, facilita mucho la depuración y pruebas y resulta sencillo crear instaladores específicos para cada plataforma.

3.6.1. Front-End

Para el desarrollo de la parte visual de la aplicación, se usará HTML5 [13] y CSS3 [14], sin utilizar ninguna tecnología adicional, ya que aunque se velará por la usabilidad de la aplicación, no se van a centrar muchos recursos en el desarrollo de la interfaz, ya que el principal objetivo del proyecto es la seguridad.

3.6.2. Back-End

Para el desarrollo de la lógica de la aplicación y su funcionamiento interno se usarán varias tecnologías. Como lenguaje conductor del desarrollo se usará JavaScript [15], que servirá para unificar los distintos componentes.

Node.js

Es un entorno de ejecución para JavaScript sobre el que está construido Electron [11]. Se construye sobre el motor V8 de Google [16] y permite ejecutar con JavaScript tareas del lado del servidor que de otra forma no sería posible.

Dispone también de varios tipos y librerías que permiten el cifrado de los datos y la seguridad de los mismos durante su manipulación.

SQLite

Se trata de un sistema de gestión de bases de datos relacionales compatible con Atomicidad, Consistencia, Aislamiento y Durabilidad (ACID) en forma de librería [17]. Esto permite que no requiera de una arquitectura cliente-servidor y se pueda integrar directamente en el programa. Además, concentra todos los datos en un único fichero facilitando la generación de respaldos de los datos.

API REST

Aunque no sea una tecnología concreta, sino una arquitectura software, cabe mencionarlo ya que será la forma de comunicación del proyecto con los distintos servicios de almacenamiento en la nube que usemos para respaldar los datos de la aplicación. Esto nos permite una gran interoperabilidad ya que la estructura para contactar con los distintos servicios seguirá el mismo patrón siempre.

Open Authorization 2 (OAuth2)

Se trata de un estándar abierto que permite autorizar de forma segura a nuestra aplicación para usar los servicios de almacenamiento en la nube del usuario sin que tenga que confiar en nuestro proyecto sus datos personales [18].

3.7. Conclusiones del análisis

Tras realizar este análisis, se considera viable el desarrollo del proyecto con el objetivo de obtener una aplicación auto contenida de gestión de contraseñas. Para su diseño y desarrollo se emplearán las tecnologías expuestas y para los métodos de cifrado se han tenido en cuenta varias consideraciones para que la información se encuentre bajo unas medidas de seguridad altas.

Considerando las recomendaciones del NIST en materia de cifrado [8] [19], se usará el algoritmo de cifrado simétrico AES con una clave de 256 bits. Para la derivación de claves se utilizará la función PBKDF2 haciendo uso de la función hash SHA con 512 bits, un salt de 128 bits y un millón de iteraciones. Según esta organización, con estas medidas de seguridad la información de nivel crítico debería estar suficientemente protegida.

4

Diseño y Desarrollo

4.1. Introducción

En este apartado se expondrán los aspectos más relevantes en cuanto al diseño de la aplicación y cómo se han implementado las distintas tecnologías durante su desarrollo. Empezaremos con un vistazo general al diseño de la aplicación y luego analizaremos por un lado el *Back-End* y por otro el *Front-End*.

4.2. Diagrama de Bloques

En el diagrama de la figura 4.1 se presenta la descripción de alto nivel del sistema. A partir de éste explicaremos el diseño y desarrollo de la aplicación en los siguientes apartados.

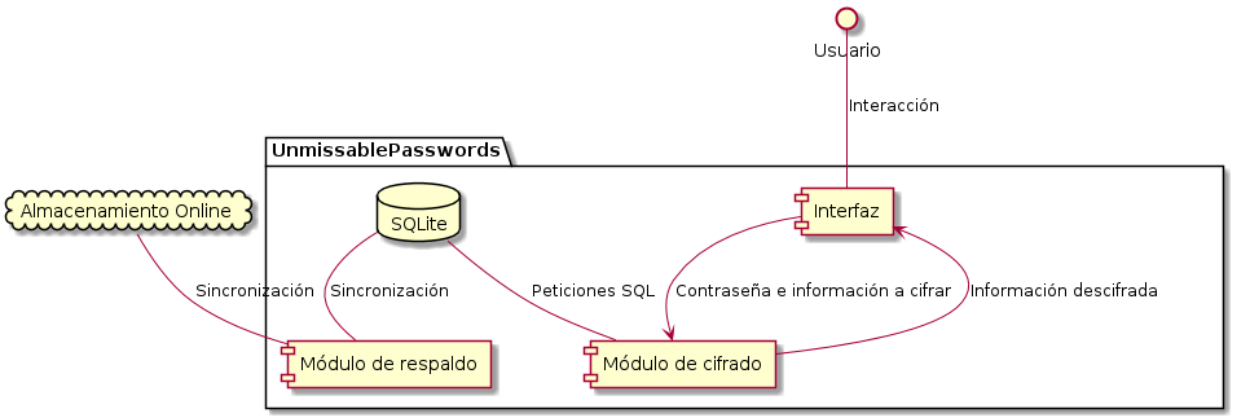

Figura 4.1: Estructura del sistema del proyecto

Toda la aplicación está integrada en el framework de Electron, en el cual se han diferenciado varios componentes y se interactúa con el usuario y los posibles servicios de la nube.

4.3. Autenticación de segundo factor

En un primer momento se planteó usar para el acceso a la aplicación, una autenticación de segundo factor junto con la contraseña para aumentar la seguridad; lo cual consiste en añadir una validación adicional a la contraseña. Sin embargo, finalmente se descartó en la fase de diseño. Esto fue debido a que por un lado, propuestas como usar una autenticación biométrica o una llave hardware escapaban de las posibilidades del proyecto y otros métodos como códigos de verificación suponían confiar en un tercero. Además, suponiendo un ataque en el que se tenga acceso a todo el sistema, sería posible sortear las medidas de seguridad a nivel de interfaz y acceder directamente al archivo con la base de datos, que se encuentra protegido únicamente por la contraseña maestra.

Por tanto, como el objetivo del proyecto era no depender de ningún agente externo para el correcto funcionamiento de la aplicación, los cuales no controlamos y pueden ser atacados, no se ha implementado una autenticación de segundo factor. No obstante, el uso de sistemas biométricos o similares que se puedan integrar como parte de la clave de cifrado junto con la contraseña como por ejemplo con la concatenación de un hash, se plantearán como trabajo futuro.

4.4. Back-End

El *Back-End* se refiere a toda la lógica y funcionamiento interno de la aplicación que no es apreciable de manera explícita por el usuario. En esta sección abordaremos los aspectos más importantes. Cabe resaltar que todo el *Back-End* del proyecto se ha realizado con JavaScript [15], más concretamente haciendo uso de Node.js [11], del que se han usado módulos y librerías para integrar toda la funcionalidad.

4.4.1. Base de Datos (SQLite)

Tecnología usada y arquitectura

Para la creación de la base de datos de la aplicación se ha hecho uso de la tecnología SQLite [17]. Esta es un sistema de gestión de bases de datos implementado en forma de librería escrita en C. Gracias a la compatibilidad del motor V8 con módulos en C++, el cual es el encargado de ejecutar Node.js, esta librería se ha podido integrar en la aplicación y hacer uso de ella.

La motivación para usarla es que no requiere de una arquitectura cliente servidor sino que simplemente se requería integrar esta librería (figura 4.2) y espacio en memoria para almacenar la base de datos como un fichero de la aplicación, lo cual permitía que la base de datos quedara auto contenida en el proyecto. Además, con el uso de la versión 3 nos permitía manejar bases de tamaño hasta 2 terabytes que para una aplicación local en el equipo de un usuario lo consideramos suficiente.

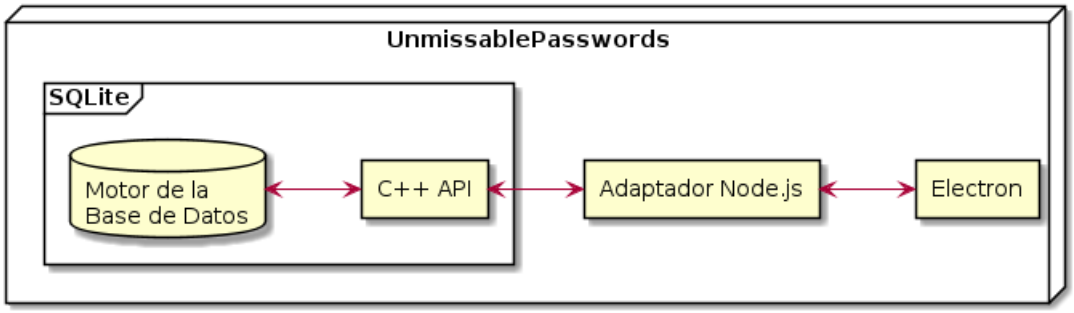

Figura 4.2: Representación de la arquitectura e integración de la base de datos

Almacenamiento y cifrado de la información

La estructura para el almacenamiento de los datos es sencilla, al iniciar la aplicación por primera vez, se crean dos tablas (Figura 4.3).

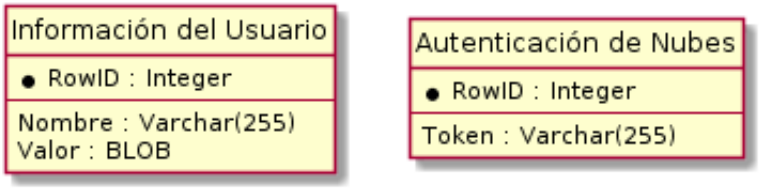

Figura 4.3: Tablas creadas para el almacenamiento de información

La primera de ellas sirve para guardar la información que requiera el usuario. Tiene tres columnas: identificador de fila, nombre de la información e información. El primero se encuentra en claro, pero cuando se hace uso de la base de datos, la información a insertar en los otros dos siempre es cifrada campo a campo. Por tanto, un atacante que acceda a la base de datos únicamente observará información inteligible.

La segunda tabla se utiliza para el almacenamiento de tokens de los servicios de almacenamiento *online* que utilice el usuario de la aplicación. Aunque solo se ha llegado a implementar la integración con Google Drive [20], se ha tomado esta decisión de diseño para en un futuro poder tener sincronizados más de un servicio. La tabla dispone de dos columnas, un identificador de fila y el campo para el token, el cual se guarda cifrado al igual que la información de la otra tabla.

Destacar que la información solamente está en claro en el momento en el que el usuario la está visualizando en la interfaz o cuando la está insertando. En la base de datos nunca es almacenada información del usuario sin cifrar. Siempre debe pasar por el módulo de cifrado.

Validación de la clave del usuario

Para validar la clave del usuario, lo que se hace es realizar una inserción en la primera fila de la tabla de información del usuario número largo cuando este inicializa la aplicación. En esta implementación se utilizan los primeros 64 dígitos de pi, cifrados con la clave maestra. Este número es conocido como *Magic Number*. De esta forma, cada vez que se inicia la aplicación se intenta descifrar la primera fila con la clave proporcionada por el usuario y se comprueba que coincida con pi, si no lo hace, la contraseña introducida no es correcta.

Con esta medida no es necesario almacenar la contraseña del usuario en ningún momento e incrementamos la seguridad de la aplicación.

Tipo de información almacenada

Por último, destacar que para almacenar la información se hace uso del tipo de datos Binary Large Object (BLOB) , lo cual permite que la base de datos no se limite únicamente a guardar cadenas de texto, sino que con una actualización de la interfaz, también se podría almacenar contenido multimedia, documentos o cualquier tipo de dato.

4.4.2. Módulo de cifrado

Para implementar los algoritmo de cifrado se ha usado el módulo *Crypto* de Node.js. Este dispone de clases y funciones que nos han permitido implementar los algoritmos tal y como se consideró en las conclusiones del análisis 3.7 [21].

Esto se ha abstraído en dos funciones, una para cifrar y otra para descifrar que requieren de dos parámetros, la contraseña maestra del usuario y la información que se quiere cifrar o descifrar. Son usadas cuando el usuario requiere que le sea mostrada información guardada en la base de datos o necesita introducir información en ella.

4.4.3. Módulo de respaldo

Este módulo se encarga de sincronizar el almacenamiento *online* con la base de datos local. Como se usa SQLite, únicamente debe encargarse de un fichero. Para el proyecto únicamente se ha implementado la sincronización con Google Drive [20], pero se ha desarrollado de forma que sea posible implementar más servicios sin mucho esfuerzo.

El protocolo de funcionamiento consiste en la operativa que se detalla a continuación. Primero, comprueba si existe o no información guardada en la base de datos local y en el almacenamiento *online*. Si existen ambos y coinciden, no realiza ninguna acción. Por el contrario, si no existe información

en el almacenamiento *online* o difiere con la información local, sobre escribe la información *online*. Si no existe información local y existe información *online*, descarga la información *online*.

Se ha tomado esta decisión siguiendo el principio del escenario en el que un atacante tuviera acceso a todos los datos, considerando el almacenamiento en la nube más vulnerable que el almacenamiento en local. Por tanto, para evitar una pérdida de datos debido a una sincronización en el que el respaldo haya sido modificado, sólo se confiará en la nube cuando no exista otra posibilidad, es decir, cuando no existan en local. En ese caso una manipulación de los datos por parte de un tercero sería irrelevante, puesto que el usuario de la aplicación se daría cuenta de ello al comprobar que la información que ha sido descargada resulta inteligible al descifrarla, pero en caso de pérdida de los datos en local, existe la posibilidad de su recuperación.

OAuth2

Para garantizar que la información sensible del usuario se vea comprometida lo mínimo posible, para sincronizar con Google Drive se utiliza el protocolo Open Authorization Version 2 (OAuth2) [18].

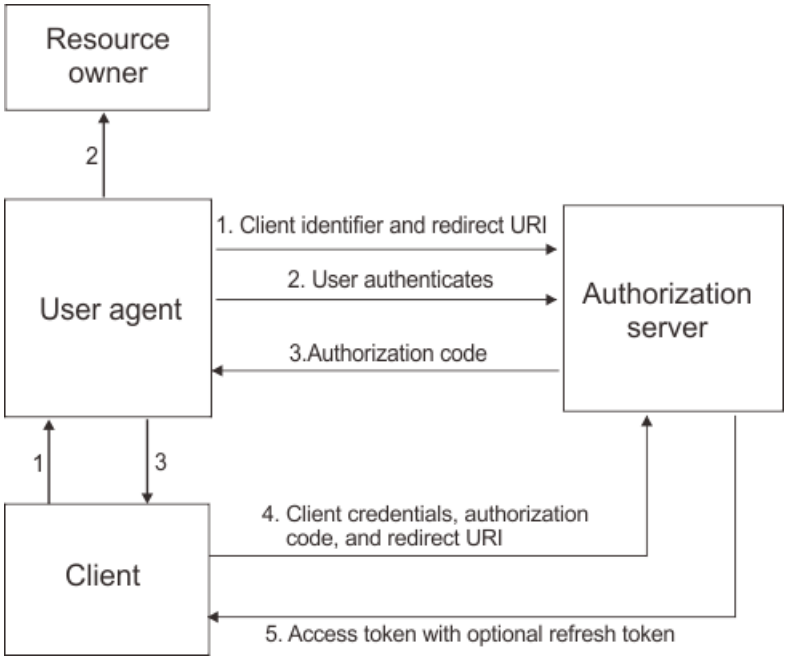

Figura 4.4: Flujo del protocolo OAuth 2.0. Fuente: IBM

Esto permite que mediante un enlace generado por Google Drive el usuario se pueda autenticar en este servicio obteniendo un *token* que la aplicación guarda en la base de datos cifrado y nos habilita para subir y descargar archivos sin que el usuario tenga que compartir su usuario y contraseña de este servicio con nosotros. Además, ese token es únicamente compatible con nuestra aplicación, por lo que un tercero no puede hacer uso del mismo.

API REST

La comunicación con Google Drive se realiza mediante *API REST*, arquitectura basada en el protocolo HyperText Transfer Protocol (HTTP) . Para ello, cada acción que queramos realizar dispone de una URL a la que dirigirse, la cual puede ser complementada con parámetros y datos en formato JSON y recibimos la respuesta como datos en notación JSON . Es importante porque esta arquitectura es independiente del lenguaje de programación y permite una interoperabilidad muy elevada.

4.5. Front-End

En esta aplicación, el *Front-End* no ha sido la mayor prioridad aunque se ha cuidado que sea sencilla y usable por parte del usuario. Se ha desarrollado principalmente en HTML5 y CSS3 con fragmentos de JavaScript para el desarrollo de los elementos dinámicos. Está integrado en su totalidad por el módulo de interfaz.

4.5.1. Interfaz

Figura 4.5: Interfaz del login de la aplicación

Esta es la parte de la aplicación con la que el usuario puede interactuar. Consta principalmente de cuatro pantallas:

- Registro
- Login
- Panel
- Vinculación de nubes

La pantalla de registro aparece cuando el usuario accede a la aplicación por primera vez y permite al usuario inicializar la base de datos con una contraseña maestra. En sucesivas ocasiones aparecerá una pantalla de login para que pueda validar su contraseña y acceder al panel. En el panel el usuario

puede gestionar su información; en un primer momento sólo se descifran los nombres de referencia de las informaciones que guarda el usuario y únicamente se descifran uno a uno los secretos en el momento en que el usuario lo requiere, pinchando sobre el nombre de cada información. Por último, desde el panel puede acceder a la vinculación de nubes para que su información sea sincronizada.

Cabe destacar también, que por motivos de seguridad, la interfaz está programada de forma que si detecta una inactividad superior a 10 segundos por parte del usuario, automáticamente cierra la aplicación para garantizar que no es usada por terceros.

En el apéndice A se puede encontrar el flujo de trabajo de un usuario para introducir datos en la aplicación.

5

Pruebas y resultados

5.1. Introducción

Al tratarse de un producto software, es necesario comprobar el correcto funcionamiento del código desarrollado y cerciorarse de su calidad y rendimiento. Por ello a lo largo de este capítulo se tratarán los aspectos relacionados con las pruebas y resultados del proyecto.

5.2. Funcionalidad

Tras la realización de las pruebas, los resultados han sido los esperados y se ha comprobado el cumplimiento de todos los requisitos establecidos para el proyecto en la fase de análisis 3.5.

Dada la dificultad de realizar pruebas sobre un lenguaje como JavaScript, la funcionalidad se ha implementado de forma incremental por módulos probando exhaustivamente cada añadido mediante pruebas de caja blanca. A ello ha ayudado las herramientas que ofrece Node.js para realizar pruebas y las herramientas de depuración de Electron. Estas pruebas de caja blanca se han realizado probando para cada funcionalidad sus posibilidades mediante scripts en para la lógica de la aplicación y mediante la consola de depuración de Chromium (integrado en Electron) para la interfaz.

Más adelante, con todos los módulos construidos, se han realizado pruebas de integración y pruebas de caja negra sobre el conjunto de la aplicación. Para ello se empezó integrando la base de datos con el módulo de cifrado, más adelante se añadió la interfaz, y por último el módulo de respaldo.

5.3. Usabilidad

Para comprobar la correcta adaptación de la aplicación de la aplicación a los usuarios, se han realizado pruebas con personas reales de distinta edad y conocimientos informáticos. Se les pidió que realizaran distintas acciones con la aplicación y después se les preguntó por su experiencia además de estimar el tiempo que les costó realizar las acciones.

Los resultados concluyen que la aplicación resulta sencilla de utilizar, debido a que la funcionalidad está muy restringida ya que su único objetivo es guardar y mostrar secretos. Sin embargo, los usuarios quisieron dejar constancia de que en las operaciones en las que era necesaria una intervención de la base de datos, la aplicación se mostraba lenta. Esto se justifica debido a que como se impuso en los requisitos, primaría la seguridad frente a la usabilidad y para poder garantizar unas fuertes medidas de seguridad, el cifrado y descifrado de información requiere entre uno y tres segundos, según se ha podido comprobar.

5.4. Compatibilidad

La compatibilidad de la aplicación, tal y como se especificó en los requisitos, debía ser posible con los tres sistemas operativos más populares de escritorio: Windows, Linux y MacOS. Gracias al uso del framework Electron, ha sido muy sencillo empaquetar la aplicación para estas tres plataformas sin encontrar inconvenientes ni problemas ya que se trata de un proceso prácticamente automático en el que únicamente hay que ejecutar un script y especificar las opciones para la compilación en un fichero de configuración en formato JSON .

Se ha podido compilar la aplicación para los tres sistemas, sin inconvenientes en el proceso. Sin embargo, sólo se ha podido probar en Windows y Linux, donde ha funcionado correctamente, ya que no se disponía de ningún dispositivo con MacOS para realizar una prueba de compatibilidad.

6

Conclusiones y trabajo futuro

6.1. Conclusiones

El resultado final de este proyecto se puede considerar satisfactorio puesto que se han conseguido los objetivos que se perseguían: desarrollar una aplicación que permita a un usuario guardar y gestionar su información de forma segura y sin la intervención de agentes externos para el correcto funcionamiento de la misma.

Además, la aplicación es capaz de realizar copias de seguridad de los datos en caso de que el usuario lo desee y proteger de esta forma la información frente a fallos del dispositivo.

El uso de algoritmos de cifrado de dominio público ha supuesto muchas garantías para la seguridad de la aplicación al estar sometidos constantemente a estudio y ha permitido ahorrar todo el tiempo que hubiera conllevado el desarrollo de un algoritmo propio, para el que realmente, no hubiera habido necesidad real.

El uso del framework Electron ha sido clave para obtener un desarrollo multiplataforma sin tener que invertir recursos específicos para cada sistema operativo en el que se quería que la aplicación fuese operativa.

La apuesta por SQLite ha permitido que el diseño de la aplicación se mantuviera simple, a diferencia de si hubiéramos implementado otras soluciones de gestión de bases de datos más complejas.

A destacar, opino que la funcionalidad conseguida permite prescindir de otras aplicaciones de gestión de secretos que guardan nuestros datos en servidores externos y no depender de una conexión a Internet para ello sin sacrificar en seguridad.

Por contra, también hay que considerar que la aplicación tiene limitaciones, como el hecho de que actualmente es monousuario y no permite actualmente almacenar información más allá de texto.

En definitiva, el objetivo del proyecto se ha cumplido con una satisfacción alta.

6.2. Trabajo Futuro

Este proyecto consistía en el desarrollo de un gestor de secretos auto contenido que garantizara la seguridad de la información del usuario.

En un futuro, se podría realizar una modificación de la misma para soportar un sistema multiusuario con el que varias personas pudieran guardar su información en la misma instancia de la aplicación.

Por otro lado, también sería interesante implementar el uso de llaves de seguridad hardware o autenticación biométrica que formaran parte de la clave de cifrado de los datos, con ánimo de mejorar la seguridad.

También sería interesante ampliar el soporte de la aplicación para que fuera ejecutable en dispositivos móviles o mejorar la interfaz para que pudiera almacenar más tipos de información.

Por tanto, teniendo en cuenta lo expuesto y más ideas que puedan ocurrir, pensamos que el proyecto tiene un amplio margen de desarrollo por el que podría expandirse.

Bibliografía

[1] “Lastpass.” [En línea]. Disponible en: LastPass. [Accedido: 18-Jun-2019].

[2] “Especificación del Estándar AES.” [En línea]. Disponible en: Especificación AES. [Accedido: 18-Jun-2019].

[3] “Especificación de PBKDF2.” [En línea]. Disponible en: Especificación PBKDF2. [Accedido: 18-Jun-2019].

[4] “DashLane.” [En línea]. Disponible en: DashLane. [Accedido: 18-Jun-2019].

[5] “1Password.” [En línea]. Disponible en: 1Password. [Accedido: 18-Jun-2019].

[6] “Password Safe.” [En línea]. Disponible en: Password Safe. [Accedido: 18-Jun-2019].

[7] “KeePassXC.” [En línea]. Disponible en: KeePassXC. [Accedido: 18-Jun-2019].

[8] “Recomendaciones del NIST en el uso de algoritmos de cifrado y longitud de claves.” [En línea]. Disponible en: Recomendaciones cifrado y claves. [Accedido: 18-Jun-2019].

[9] “Recomendaciones del NIST para el uso de los modos de cifrado de bloques.” [En línea]. Disponible en: Recomendaciones Cifrado de Bloques. [Accedido: 18-Jun-2019].

[10] “Electron.” [En línea]. Disponible en: Electron. [Accedido: 18-Jun-2019].

[11] “Node.js.” [En línea]. Disponible en: Node.js. [Accedido: 18-Jun-2019].

[12] “Chromium.” [En línea]. Disponible en: Chromium. [Accedido: 18-Jun-2019].

[13] “HTML5.” [En línea]. Disponible en: HTML5. [Accedido: 18-Jun-2019].

[14] “CSS3.” [En línea]. Disponible en: CSS3. [Accedido: 18-Jun-2019].

[15] “JavaScript.” [En línea]. Disponible en: JavaScript. [Accedido: 18-Jun-2019].

[16] “Motor V8.” [En línea]. Disponible en: V8. [Accedido: 18-Jun-2019].

[17] “SQLite.” [En línea]. Disponible en: SQLite. [Accedido: 18-Jun-2019].

[18] “Especificación e información sobre el estándar OAuth 2.0.” [En línea]. Disponible en: OAuth 2.0. [Accedido: 18-Jun-2019].

[19] “Recomendaciones del NIST para el uso de funciones de derivación de clave.” [En línea]. Disponible en: Recomendaciones PBKDF2. [Accedido: 18-Jun-2019].

[20] “Google Drive API.” [En línea]. Disponible en: Google Drive API. [Accedido: 18-Jun-2019].

[21] B. Schneier, *Applied Cryptography*. John Wiley & Sons, 1993.

Definiciones

Dark Web Contenido público de la World Wide Web que existe en *darknets*, redes que se superponen al Internet público y requieren de software específico, configuraciones o autorización para acceder.

framework Entorno o marco de trabajo que engloba una serie de conceptos, prácticas y criterios estandarizados para afrontar un tipo de problemática particular.

hash Salida de un algoritmo matemático que transforma cualquier bloque arbitrario de datos en una nueva serie de caracteres con una longitud fija.

Linux Sistema operativo libre de tipo Unix creado por Linus Torvalds.

MacOS Sistema operativo de escritorio desarrollado por Apple.

MySQL Sistema de gestión de bases de datos relacional muy popular en sistemas web desarrollado por Oracle.

Open Source Modelo de desarrollo de software basado en la colaboración abierta.

salt Conjunto de bits aleatorios que se usan como parte de la entrada para una función de derivación de claves.

script Archivo de procesamiento por lotes.

Windows Sistema operativo de escritorio desarrollado por Microsoft.

Acrónimos

2FA Autenticación de segundo factor.

ACID Atomicidad, Consistencia, Aislamiento y Durabilidad.

AES Advanced Encryption Standard.

AWS Amazon Web Services.

BLOB Binary Large Object.

CBC Cipher-block chaining.

CSS3 Cascade Style Sheet Version 3.

ECB Electronic Code-Book.

HMAC Código de Autenticación de Mensajes en clave-hash.

HTML5 HyperText Markup Languaje Version 5.

HTTP HyperText Transfer Protocol.

JSON JavaScript Object Notation.

NIST Instituto Nacional de Normas y Tecnología.

OAuth2 Open Authorization Version 2.

PBKDF2 Password-Based Key Derivation Function 2.

SHA Secure Hash Algorithm.

SSH Secure Shell.

TFG Trabajo de Fin de Grado.

Apéndices

A

Flujo de trabajo de un usuario para registrar datos

En este anexo se mostrará visualmente cómo debe proceder un usuario para insertar datos en la aplicación.

En primer lugar, al iniciar la aplicación debe introducir la contraseña (Figura A.1).

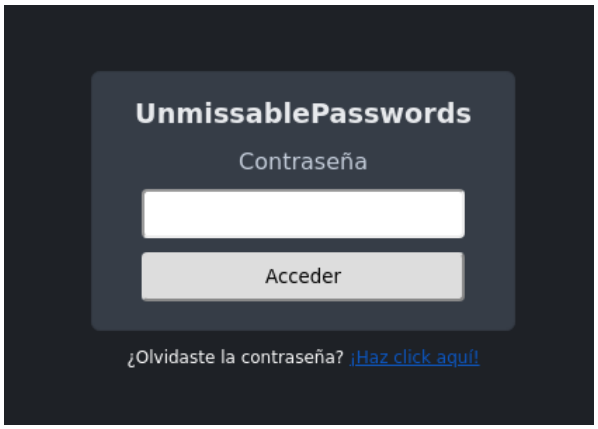

Figura A.1: Inicio de sesión.

A continuación, debe pulsar el botón para añadir datos (Figura A.2).

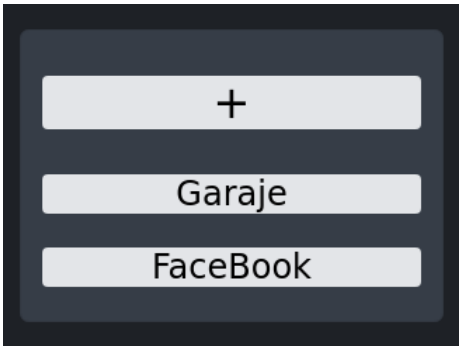

Figura A.2: Panel de control de la aplicación.

Entonces se desplegará un pequeño formulario donde puede introducir la información que desee guardar (Figura A.3).

Figura A.3: Añadiendo datos a la aplicación.

Al enviarlo, los datos quedarán añadidos al sistema (Figura A.4).

Figura A.4: Panel con los nuevos datos.

Por último, si el usuario pincha sobre el nombre de los datos, se descifra y muestra el valor (Figura A.5).

Figura A.5: Se muestra la información solicitada.